

Power Efficient and Fast Updating File Approach in Wireless Grids

Mr. K. SUPEKH PUMAK¹, Dr. Y. RAJASREO RAI², Dr. K. MANJUMETHACHARI³

¹Electronics & Communication Engineering, SSJ Engineering College, Hyderabad, India.

²Electronics & Communication Engineering, SRIDEVI WOMEN'S Engineering College, Hyderabad, India.

³Electronics & Communication Engineering, GITAM University, Hyderabad campus, India.

ABSTRACT— Ternary Content-Addressable Memories (TCAMs) are becoming popular for designing high-throughput forwarding engines on routers. They are fast, cost-effective and simple to manage. However, a major drawback of TCAMs is their large power consumption. This paper presents architectures and algorithms for making TCAM-based routing table more power efficient. The proposed architecture and algorithms are simple to implement, use commodity TCAMs, and provide worst-case power consumption guarantees (independent of routing table contents). The most existing TCAM-based forwarding engines involve shifting TCAM entries when the forwarding table is updated, typically incurring a lengthy update duration. And also proposed a TCAM-based longest prefix forwarding engine with fast updating. The key idea behind the design is to maintain the forwarding table in a TCAM according to the Minimum Independent Prefix Set.

Key words: TCAM, Delay, Network algorithm, memory architecture, bit selection, prefixes.

I. INTRODUCTION

Ternary Content Addressable Memories (TCAMs) are fully associative memories that allow a “don’t care” state to be stored in each memory cell in addition to 0s and 1s. This feature makes them particularly attractive for packet classification and route lookup applications which require longest prefix matches. When a destination address is presented to the TCAM, each TCAM entry is looked up in parallel, and the longest prefix that matches the address is returned. Thus, a single TCAM access is sufficient to perform a route lookup operation. In contrast, conventional ASIC-based designs that use tries may require multiple memory accesses for a single route lookup. Therefore, routing latencies for TCAM-based routing tables are significantly lower than ASIC-based tables. Moreover, TCAM-based tables are typically much easier to manage and update than tables implemented using tries. Despite these advantages, routing vendors have been slow in adopting TCAM devices in packet forwarding engines because of two main reasons. First, TCAM devices have traditionally been more expensive and less dense compared to conventional ASIC-based devices. However, both the density and the cost of TCAMs have dramatically improved in the past few years, making them a viable alternative to ASIC-based designs in high-speed core routers. The second reason is that of high power consumption. Current high-density TCAM devices

consume as much as 12–15Watts each when all the entries are enabled for search. Moreover, a single line card may require multiple TCAMs to handle filtering and classification as well as IP lookup on large forwarding tables. This high power consumption number affects costs in two ways—first; it increases power supply and cooling costs that account for a significant portion of an ISP’s operational expenses [1]. Second, it reduces port density since higher power consumption implies that fewer ports can be packed into the same space (e.g., router rack) due to cooling constraints. Therefore, it is important to minimize the power budget for TCAM-based forwarding engines to make them economically viable. In this paper, we focus on the problem of making TCAM-based forwarding engines more power efficient by exploiting commonly available TCAM features. Several TCAM vendors (e.g., [3]) now provide mechanisms for searching only a part of the TCAM device in order to reduce power consumption during a lookup operation. We take advantage of this feature to provide two different power efficient TCAM based architectures for IP lookup. Both of our architectures utilize a two stage lookup process. The basic idea in either case is to divide the TCAM device into multiple partitions (depending on the power budget). When a route lookup is performed, the results of the first stage lookup are used to selectively search only one of these partitions during the second stage lookup. The two architectures differ in the mechanism for performing the

first stage lookup. In the first architecture, we use a subset of the destination address bits to hash to a TCAM partition (the *bit-selection* architecture), allowing for a very simple hardware implementation. The selected bits are fixed based on the contents of the routing table. In the second architecture, a small trie (implemented using a separate, small TCAM) is used to map a prefix of the destination address to one of the TCAM partitions in the next stage (the *trie-based* architecture). This adds some design complexity, but we show that it results in significantly better worst-case power consumption.

II. TCAMS FOR ADDRESS LOOKUPS

A Ternary Content Addressable Memory (TCAM) is a fully associative memory that allows a “don’t care” state for each memory cell, in addition to a 0 and a 1. A memory cell in a “don’t care” state matches both 0s and 1s in the corresponding input bit. The contents of a TCAM can be searched in parallel and a matching entry, if it exists, can be found in a single cycle (using a single TCAM access). If multiple entries match the input, the entry with the lowest address in the TCAM is typically returned as the result. The characteristics described above make TCAMs an attractive technology for IP route lookup operations where the destination address of an incoming packet is matched with the *longest matching prefix* in a routing table database. TCAMs can be used to implement routing table lookups as follows. If the maximum prefix length is W , then each routing prefix of length n ($\leq W$) is stored in the TCAM with the rightmost $W - n$ bits as “don’t cares”. For example, the IPv4 prefix 192.168.0.0/15 will have “don’t care” in the last 17 bit positions. To ensure that the longest prefix match is returned, the prefixes in the TCAM must be sorted in order of decreasing prefix length. The sorting requirement makes it difficult to update the routing table. However, recent work [9] has proposed innovative algorithms for performing TCAM updates simply and efficiently. As mentioned earlier, the two main disadvantages of using TCAMs have traditionally been the high cost to density ratio and the high power consumption. Recent developments in TCAM technology have effectively addressed the first issue is TCAM devices with high capacity (up to 18Mbits) and search rates of over 100 Million lookups/second [3], [8] are now coming to market with costs that are competitive with alternative technologies. The power consumption issue still remains somewhat unresolved. The main component of power consumption in TCAMs is proportional to the number of searched entries. The growth trends in the routing tables in the

Internet core [2] have prompted routing vendors to design routing engines capable of scaling up to 1 million entries. TCAM vendors today have started providing mechanisms that can reduce power consumption by selectively addressing smaller portions of the TCAM. Each portion (called a sub-table or database) is defined as a set of TCAM blocks. A TCAM block is a contiguous, fixed-sized chunk of TCAM entries, usually much smaller than the size of the entire TCAM. Currently, TCAMs typically support a small number of sub-tables (such as 8 sub-tables addressed by a 3-bit ID), but the same mechanism could be used to support more sub-tables. Typically, each sub-table is intended for use in a different lookup/classification application. In this paper, we exploit the mechanism described above to reduce power consumption for route lookup applications. Given that the power consumption of a TCAM is linearly proportional to the number of searched entries, we use this number as a measure of the power consumed. Clearly, if the TCAM is partitioned into K equal-sized sub-tables, it is possible to reduce the maximum number of entries searched per lookup operation to as low as $1/K$ of the TCAM size. However, this raises three important issues. First, we need to partition the TCAM into sub-tables. Second, given an input, we need to select the right partition and search it. Finally, for a given partitioning scheme, we need to compute the size of the largest partition over all possible routing tables, so that hardware designers can allocate a power budget.

III. FORWARDING ENGINE ARCHITECTURE

The system architecture of our proposed MIPS Forwarding Engine (MIPS-FE) is depicted in Fig.1. Besides TCAM and its associated SRAM used respectively to accommodate the forwarding prefixes and their corresponding next-hops, there are two other major components: data plane and control plane, in support of forwarding table lookups and updating.

A. Data plane

The data plane takes the IP header of a received packet and passes it to TCAM, where the MIPS forwarding table is searched, with the address of the matched entry delivered to SRAM for retrieving the proper next-hop field. Finally, SRAM returns the next-hop result back to the data plane.

B. Control plane

The control plane translates a received update message into some update operations via an auxiliary 1-

bit trie structure maintained therein and passes them to TCAM for incremental updates.

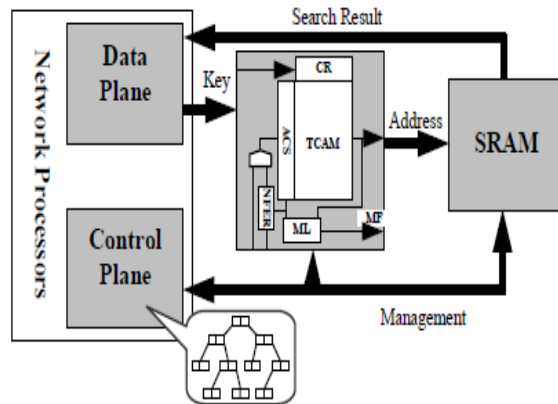


Fig 1. System architecture of TCAM

C. TCAM

Although working with conventional 2-port TCAM, our design is described here under TCAM of 3 ports (with two for lookups and one for updating/maintenance). A dedicated TCAM updating port prevents table maintenance from interrupting lookup operations. Besides, 3-port configuration allows the control plane to be separated from the data plane, operating independently without any performance penalty when carrying out MIPS table maintenance such as incremental updates and data compression.

D. SRAM

SRAM is used to accommodate next-hop information corresponding to each forwarding prefix in TCAM. It is indexed by the location of the matched prefix in TCAM and returns next-hop information kept in the entry back to the data plane as the search result. Most of these features are commonly supported by TCAM-based search engine products available in the market.

1) In addition to an inbound data port from the data plane and an outbound address port to next-hop memory, TCAM is equipped with a third port operating with the control plane. That separate port is dedicated to TCAM maintenance, permitting table updates to proceed without interrupting or suspending the search path. 2) Each TCAM entry can be tagged with *Access Control Status* (ACS) that can be any one of “invalid”, “valid”, “hit”, or “updating”. All empty and outdated entries are set to the “invalid” status. Invalid entries are not involved in the search operations (for saving power) but are allowed to be overwritten immediately. Accordingly, the deletion operation is nothing but setting a specified entry to “invalid”. All “valid” entries, if not disabled, are compared in parallel during search operation and allowed

to be updated. When a prefix in a “valid” entry is matched, the entry is set to “hit”, disallowing it to be deleted or updated until the address of its corresponding next-hop memory location is latched and then its status is reset to “valid”. This kind of hits is called “exact” hits. When an entry is under updating, it is in the “updating” status. When an “updating” entry is hit, its corresponding next-hop field is not returned until the update completes. This kind of hits is referred to as “suspended” hits. 3) During a search operation, the search key (an IP address) is placed in a special register, named the *Compare and Register* (CR). When a “suspended” hit occurs, the key retains in CR until the updating operation finishes. When a new search starts, its key is loaded to CR. 4) If a hit happens, no matter an *exact* or a *suspended* hit, a *Match Flag* (MF) is asserted to announce that the data in CR is found in memory. The *Match Flag* is reset until the search operation completes, i.e., the address of its corresponding next-hop memory location is latched. 5) A *Next Free Entry Register* (NFER) is used to keep the address of a TCAM location that is available for accommodating a new prefix. Specifying NFER is simple because the absence of the order constraint in TCAM eliminates the need to rearrange TCAM entries after each deletion operation.

IV. THE BIT SELECTION ARCHITECTURE

In this section, we describe the bit selection architecture for TCAM based packet forwarding engines. The core idea here is to split the entire routing table stored in the TCAM device into multiple sub-tables or *buckets*, where each bucket is laid out over one or more TCAM blocks. Each route lookup is now a two-stage operation where a fixed set of bits in the input is used to hash to one of the buckets. The selected bucket is then searched in the second stage. The hashing is performed by some simple glue logic placed in front of the TCAM device (which we refer to as the *data TCAM*). We restrict the hash function here to be such that it simply uses the selected set of input bits (called the *hashing bits*) as an index to the appropriate TCAM bucket. This bound is dependent on the size of the routing table and is proportional to the maximum number of blocks searched for any lookup. We then describe some heuristics to efficiently split a given routing table into buckets and how to map these buckets onto TCAM blocks.

A. Forwarding engine architecture

The forwarding engine design for the bit selection architecture is based on a key observation made in a recent study [2] of routing tables in the Internet core. This study pointed out that a very small percentage of the

prefixes in the core routing tables (less than 2% in our datasets) are either very short (< 16 bits) or very long (>24 bits). We therefore developed an architecture where the very short and very long prefixes are grouped into the minimum possible number of TCAM blocks. These blocks are searched for every lookup. The remaining 98% of the prefixes that are 16 to 24 bits long are partitioned into buckets, one of which is selected by hashing for every lookup. The bit-selection architecture is shown in Figure 2. The TCAM blocks containing the very short and very long prefixes are not shown explicitly. The bit-selection logic in front of the TCAM is a set of muxes that can be programmed to extract the hashing bits from the incoming packet header and use them to index to the appropriate TCAM bucket. The set of hashing bits can be changed over time by reprogramming the muxes. First, we only consider the set of 16 to 24 bit long prefixes (called the *split set*) for partitioning. Second, it is possible that the routing table will span multiple TCAM devices, which would then be attached in parallel to the bit selection logic. However, each lookup would still require searching a bucket in a single TCAM device. Third, we assume that the total number of buckets $K = 2^k$ is a power of 2. Then, the bit selection logic extracts a set of k hashing bits from the packet header and selects a prefix bucket. This bucket, along with

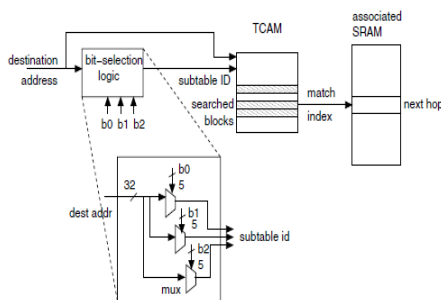


Fig.2. Forwarding engine architecture for using bit selection to reduce power consumption.

The 3 hashing bits here are selected from the 32-bit destination address by setting the appropriate 5-bit values for b_0 , b_1 and b_2 . The TCAM blocks containing the very short and very long prefixes are then searched. The two main issues now are how to select these k hashing bits, and how to allocate the different buckets among the various TCAM blocks. The first issue leads to our final assumption is we restrict ourselves to choosing the hashing bits from the first 16 bits, which is the minimum length of a prefix in the split set. The “best” hash function (that is, set of hashing bits) is the one that minimizes the size of the biggest resulting bucket.

B. The Bit Selection Heuristics

The bound on the worst case input helps designers to determine the power budget. Given such a power budget, and a routing table, it is sufficient to ensure that the set of selected hashing bits produces a split that does not exceed the power budget. We call such a split a *satisfying split*. Note that it is possible that for the given routing table, a different partitioning (with lower power consumption) exists but we only care about keeping the power consumption below the power budget. In this section, we describe three different heuristics for choosing the set of hashing bits. We then show how these heuristics can be combined to ensure that the power budget computed by Theorem. Our first heuristic is the simplest (the *simple* heuristic) and requires no computation. This is based on the following observation. For almost all the routing table traces that we have analyzed, the rightmost k bits from the first 16 bits provide a satisfying split. However, this may not be true for tables that we have not examined or for tables of the future. Therefore, better schemes may be required if these hashing bits do not yield a satisfying split. The second heuristic requires the most computation, it uses a brute force search to check all possible subsets of k bits from the first 16 bits and selects the first hashing set that satisfies the power budget. Obviously, this method is guaranteed to find a satisfying split. Since this method

compares $\binom{16}{k}$ possible sets of k bits, its running time is maximum for $k = 8$. Finally, the third heuristic is a greedy algorithm that falls between the brute force heuristic and the simple heuristic in terms of computation as well as accuracy. It may not find a satisfying split always, but has a higher chance of succeeding than the simple heuristic. To select k hashing bits, the greedy algorithm performs k iterations, selecting 1 hashing bit per iteration. Thus, the number of buckets (partitions of the routing table) doubles in each iteration. The goal in each iteration is to select a bit that minimizes the size of the biggest bucket produced by the 2-way split in that iteration. We now outline a scheme that combines each of the three heuristics to minimize the running time of the bit-selection procedure. Let M be the lower bound on the worst-case size of the largest bucket and T be the size of the entire TCAM. In addition, let P be the power consumption of the TCAM when all the entries are searched. Then the worst-case power budget is given by $P_b = (1+\alpha) \cdot M/T$. P provides a small additional margin for slack (say, 5%). It is possible to maintain a power budget of P_b using the following steps.

- 1) Split the routing prefixes using the last k of their first 16 bits. If this produces a satisfying split, stop.

2) Otherwise, apply the greedy heuristic to find a satisfying split using k hashing bits. If this produces a satisfying split, stop.
3) Otherwise, apply the brute force heuristic to find a satisfying split using k hashing bits. We remind the reader that the algorithm described above must be applied whenever route updates change the prefix distribution in the routing table such that the size of the largest bucket exceeds M . For real tables, the expectation is that such recompilations will not be necessary very often.

C. Experimental results

In this subsection, we present experimental results of applying the bit selection heuristics described in above. We evaluated the heuristics with respect to two metrics—the running time of the heuristic, and the quality of the splits produced by the heuristics. For this purpose, we applied the heuristics to multiple real core routing tables,

```

B = {};
bins = {P};
for i = 1 to k
    minmax = ∞;
    foreach bit b ∈ {1, ..., 16} - B
        binsb = {sb=0, sb=1 | s ∈ bins};
        maxb = max (binsb);
        if (minmax > maxb) then
            min_bit = b;
            minmax = maxb;
        endif
    endforeach
    B = B ∪ min_bit;
    bins = binsmin_bit
endfor

```

The above algorithm greedy algorithm for selecting k hashing bits for a satisfying split. B is the set of bits selected, and P is the set of all prefixes in the routing table. Here $s_{b=j}$ denotes the subset of prefixes in set s that have a value of j ($j = 0$ or 1) in bit position b and we present the results for 2 of those tables.

TABLE I. The two core routing tables used to test the bit selection schemes.

| Site | Location | Date | Table in Size |
|--------|----------|------------|---------------|
| rrc04 | Geneva | 11/01/2001 | 109,600 |
| oregon | Oregon | 05/01/2002 | 121,883 |

Details of these routing tables are listed in Table I. The results of applying the algorithms to the other core routing tables were similar.

Running Times: The running times for the brute force and the greedy heuristics are described. All the experiments were run on a 800 MHz PC and required less than 1MB of memory. We first consider the running time of the brute force heuristic. For the real routing tables, there were less than 12,000 unique combinations of the first 16 bits for the 16-24 bit prefixes. The running time for the brute force algorithm was less than 16 seconds for selecting up to 10 hashing bits. To explore

the worst case running times for 1M prefixes, we generated a synthetic table that has approximately 1 million prefixes with 216 unique combinations of the first 16 bits. This table was constructed by randomly picking the (non-zero) number of prefixes that share each combination of the first 16 bits. In this case, the running time can go as high as 80 seconds for selecting 8 hashing bits. Looking at the numbers for the greedy heuristic, we find that for real tables, it can run in as low as 0.05 seconds (up to 10 hashing bits) and takes about 0.22 seconds for the worst case synthetic input. This is an order of magnitude faster than the brute force heuristic. However, if the routing updates do not require frequent reorganizations of the routing tables, the brute force method might also suffice.

Quality of Splits: We now explore the nature of the splits produced by each of the three heuristics. Let N denote the number of 16-24 bit prefixes in the table, and c_{max} denote the maximum bucket size. The ratio N/c_{max} are a measure of the quality (evenness) of the split produced by the hashing bits. In particular, it is the factor of reduction in the portion of the TCAM that needs to be searched. Figure 3 & 4 shows a plot of N/c_{max} versus the number of hashing bits k . From the below figure,

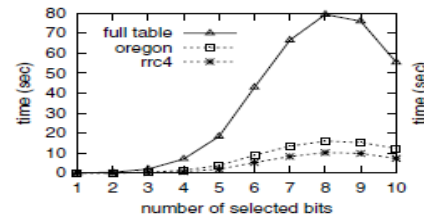


Fig.3. Running times of the brute force and greedy algorithms.

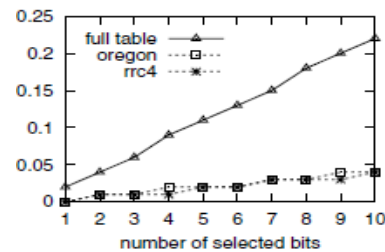


Fig.4. Running times of the brute force and greedy algorithms.

The brute force algorithm performs an exhaustive search to find the best bits to select, while the greedy algorithm may find a suboptimal set of bits. “full” is the synthetic table, while rrc4 and oregon are real core routing tables.

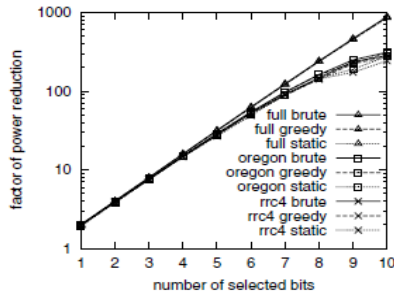


Fig 5. Power reduction factor (= size of entire prefix table/size of largest bucket) plotted on a log scale, using the different algorithms. “brute” uses the brute force method, “greedy” uses the greedy algorithm, while “static” uses the last few consecutive bits out of the first 16 bits of a prefix. We see that the ratio N/C_{max} for the brute force and greedy schemes is nearly 53 at $k = 6$; for the static scheme this ratio is around 49, while this ratio for the best possible split (a completely even split) would be $2^k = 64$. The differences between the three bit selection heuristics widens as more hashing bits are used. Since the synthetic table was generated by selecting the number of prefixes for each combination of the first 16 bits uniformly at random, it is easier to find good hashing bits for it. Hence all the three bit selection schemes provide splits that are close to ideal for the synthetic table. In contrast, real tables are less uniform than the synthetic table yielding more uneven splits, and therefore, lower power reduction ratios.

Laying out buckets on TCAM blocks: We now consider the problem of laying out the buckets (corresponding to a satisfying split) on the TCAM blocks. First, the blocks containing the very long and very short prefixes are placed in the TCAM at the beginning and the end, respectively. This ensures that the longest prefix is selected in the event of multiple matches. We now focus on the buckets containing the 16-24 bit prefixes. Let the size of the largest bucket be C_{max} , and let the size of each TCAM block be s . ideally, we would like at most $\lceil C_{max}/s \rceil$ blocks be too searched when any address is looked up. However, it is possible to show that for any TCAM with capacity N and block size s , there exists a possible split of N prefixes into buckets (of maximum size C_{max}) such that every possible layout scheme will have to lay out at least one bucket over $(\lceil C_{max}/s + 1 \rceil)$ TCAM blocks. Our scheme lays out the buckets sequentially in any order in the TCAM, ensuring that all the prefixes in one bucket are in contiguous locations. It is possible to show that for this scheme, each bucket of size c occupies no more than $\lceil c/s + 1 \rceil$ TCAM blocks. Consequently, at most $\lceil C_{max}/s + 1 \rceil$ TCAM blocks need to be searched during any lookup. Thus, our layout scheme is optimal in the sense that it matches the lower

bound discussed in the previous paragraph. The actual power savings ratio will be lower than the metric N/C_{max} plotted in Figure 5. This is because the bucket layout scheme may round up the number of searched blocks and the extra blocks containing the long and short prefixes need to be searched for every lookup. For example, consider the task of laying out a 512K-entry prefix table into a 512K-entry TCAM with 64 8K blocks. Suppose that the very short (< 16 -bit) and very long (> 24 -bit) prefixes fit into 2 blocks, while the biggest bucket contains 12K 16-24 bit prefixes. The metric N/C_{max} has the value $512K/12K = 42.67$. However, our layout scheme guarantees that the maximum number of blocks searched during a lookup would be $(\lfloor 12K/8K \rfloor + 1) + 2 = 5$, which reduces power consumption by a factor of $64/5 = 12.8$. For a TCAM with a maximum power rating of 15Watts, this results in a power budget of under 1.2 Watts, which is in the same ballpark as the SRAM based ASIC designs [3].

V. Conclusion

The bit selection architecture provides a straightforward technique for reducing the power consumption of data TCAMs. In particular, the additional hardware required for bit extraction and hashing is a set of simple muxes and can be very cost effective. However, the technique has some drawbacks. First, the worst-case power consumption guaranteed by this method is fairly high. In practice (i.e., for real tables), we saw that our heuristics provide significantly lower power consumption. For example, for a table with $N=1M$ prefixes, the worst-case analysis guarantees a power reduction ratio $N/C_{max} = 2.62$ using 3 hashing bits (from Table I), while our experimental results indicate power reduction ratios over 7.5 (from Figure 5). However, for a hardware designer who allocates a power budget, the worst-case power requirement is required to provide a guaranteed-not-to-exceed power budget. Thus, for the bit selection architecture, the designer would *be forced* to design for much higher worst-case power consumption than will ever be seen in practice. Second, the method of bit-selection described here assumes that the bulk of the prefixes lie in the 16-24 bit range². This assumption may not hold in the future. In particular, the number of long (> 24 -bit) prefixes may increase rapidly.

REFERENCES

- [1] A. McAuley, and P. Francis, “Fast Routing Table Lookup Using CAMs,” *Proc. IEEE Infocom*, vol. 3, pp. 1382–1391, March/April 1993.
- [2] V.C. Ravikumar, R. N. Mahapatra, “TCAM Architecture for IP Lookup Using Prefix Properties,” *IEEE Micro*, vol. 24, no. 2, pp. 60–69, March/April 2004.

International Journal of Computer Networks and Distributed Computing
Vol. 1, Issue 1

- [3] F. Zane, G. Narlikar, and A. Basu, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines," *Proc. IEEE Infocom*, vol. 1, pp. 42–52, March/April 2003.
- [4] Mohammad J. Akhbarizadeh, Mehrdad Nourani, Deepak S. Vijayasarathi, Poras T. Balsara. "PCAM: A Ternary CAM Optimized for Longest Prefix Matching Tasks," *Proc. 2004 IEEE Int'l Conf. on Computer Design (ICCD'04)*, pp. 6–11, Oct. 2004.
- [5] V.C. Ravikumar, R. N. Mahapatra, L. N. Bhuyan. "EaseCAM: An Energy and Storage Efficient TCAM-Based Router Architecture for IP Lookup," *IEEE Transactions on Computers*, vol. 54, no. 5, pp. 521–533, May 2005.
- [6] Anthony Gallo. Meeting Traffic Demands with Next-Generation Internet Infrastructure. *Lightwave*, 18(5):118–123, May 2001. Available at http://www.siliconaccess.com/news/Lightwave_may_01.html.
- [7] G. Huston. Analyzing the Internet's BGP Routing Table. *The Internet Protocol Journal*, 4, 2001.
- [8] IDT. <http://www.idt.com/products/>.
- [9] M. Kobayashi, T. Murase, and A. Kuriyama. A Longest Prefix Match Search Engine for Multi-Gigabit IP Processing. In *Proceedings of the International Conference on Communications (ICC 2000)*, pages 1360–1364, New Orleans, LA, 2000.
- [10] H. Liu. Routing Table Compaction in Ternary CAM. *IEEE Micro*, 22(1):58–64, January–February 2002.
- [11] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP Misconfiguration. In *Proceedings of SIGCOMM '02*, Pittsburgh, PA, August 2002.
- [12] A. J. McAuley and P. Francis. Fast Routing Table Lookup Using CAMs. In *Proceedings of Infocom '93*, pages 1382–1391, San Francisco, CA, March 1993.
- [13] Netlogic Microsystems <http://www.netlogicmicro.com>.
- [14] D. Shah and P. Gupta. Fast Updating Algorithms for TCAMs. *IEEE Micro*, 21(1):36–47, January–February 2001.
- [15] V. Srinivasan and G. Varghese. Fast Address Lookups Using Controlled Prefix Expansion. *ACM Transactions on Computer Systems*, 17(1):1–40, February 1999.